

Artificial Intelligence Nanodegree Projects

Implementing a Planning Search

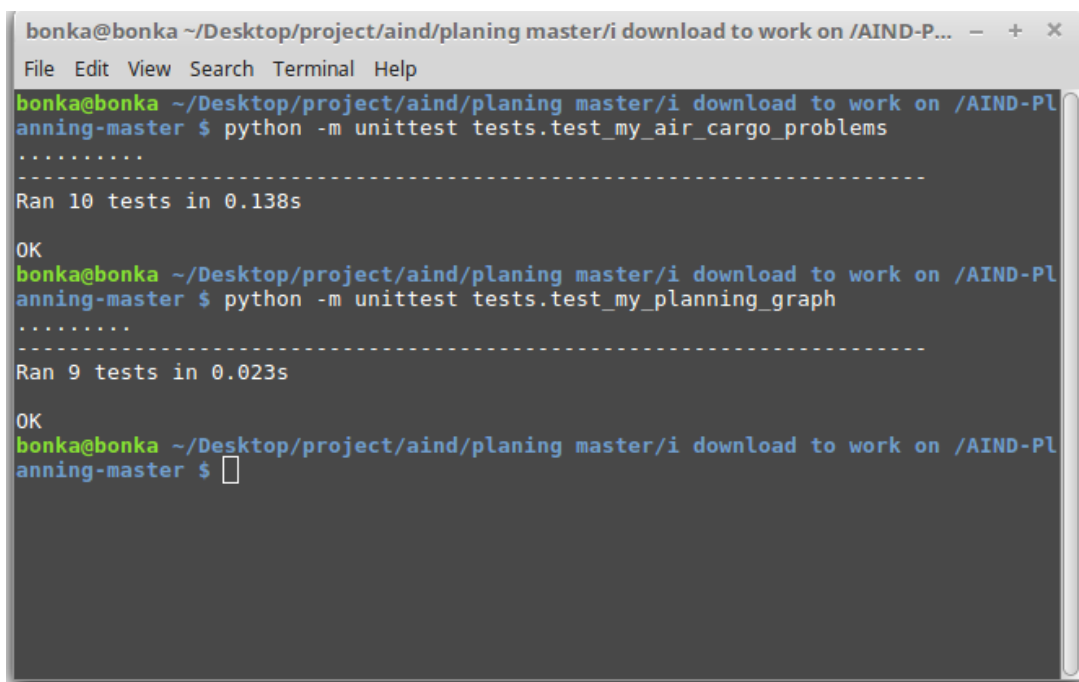
This project is an implementation of planning search agent to solve planning problems for an Air Cargo transport system. The planning graph has been used to experiment the uninformed heuristic strategy and the informed heuristic strategy and the results is tabulated below in each problem. There is comparison between the two strategies and the best has been chosen as the best heuristic for solving air cargo problems. The project start with implementation of my_planning_graph and my_air_cargo_problems.

The following commands is run on the command line to test the implementation if its corrects.

```
(a) $ python -m unittest tests.test_my_air_cargo_problems
```

```
(a) $ python -m unittest tests.test_my_planning_graph
```

Visualisation of the tests.



```
bonka@bonka ~/Desktop/project/aind/planning master/i download to work on /AIND-P... - + x
File Edit View Search Terminal Help
bonka@bonka ~/Desktop/project/aind/planning master/i download to work on /AIND-Pl
anning-master $ python -m unittest tests.test_my_air_cargo_problems
.....
-----
Ran 10 tests in 0.138s

OK
bonka@bonka ~/Desktop/project/aind/planning master/i download to work on /AIND-Pl
anning-master $ python -m unittest tests.test_my_planning_graph
.....
-----
Ran 9 tests in 0.023s

OK
bonka@bonka ~/Desktop/project/aind/planning master/i download to work on /AIND-Pl
anning-master $
```

Planning Problems

We were given three planning problems in the Air Cargo domain that use the same action schema:

Action(Load(c, p, a),

PRECOND: $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$

EFFECT: $\neg \text{At}(c, a) \wedge \text{In}(c, p)$)

Action(Unload(c, p, a),

PRECOND: $\text{In}(c, p) \wedge \text{At}(p, a) \wedge \text{Cargo}(c) \wedge \text{Plane}(p) \wedge \text{Airport}(a)$

EFFECT: $\text{At}(c, a) \wedge \neg \text{In}(c, p)$)

Action(Fly(p, from, to),

PRECOND: $\text{At}(p, \text{from}) \wedge \text{Plane}(p) \wedge \text{Airport}(\text{from}) \wedge \text{Airport}(\text{to})$

EFFECT: $\neg \text{At}(p, \text{from}) \wedge \text{At}(p, \text{to})$)

The three problems have different initial states, goals, and optimal solution length, as shown below:

- Problem 1 initial state and goal:

Init($\text{At}(C1, \text{SFO}) \wedge \text{At}(C2, \text{JFK})$
 $\wedge \text{At}(P1, \text{SFO}) \wedge \text{At}(P2, \text{JFK})$
 $\wedge \text{Cargo}(C1) \wedge \text{Cargo}(C2)$
 $\wedge \text{Plane}(P1) \wedge \text{Plane}(P2)$
 $\wedge \text{Airport}(\text{JFK}) \wedge \text{Airport}(\text{SFO})$)
Goal($\text{At}(C1, \text{JFK}) \wedge \text{At}(C2, \text{SFO})$)

The goal above can be reached using different plans, but the optimal plan length is 6 actions. Below is a sample plan with optimal length:

Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P1, SFO, JFK)
Fly(P2, JFK, SFO)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)

- Problem 2 initial state and goal:

Init($\text{At}(C1, \text{SFO}) \wedge \text{At}(C2, \text{JFK}) \wedge \text{At}(C3, \text{ATL})$
 $\wedge \text{At}(P1, \text{SFO}) \wedge \text{At}(P2, \text{JFK}) \wedge \text{At}(P3, \text{ATL})$
 $\wedge \text{Cargo}(C1) \wedge \text{Cargo}(C2) \wedge \text{Cargo}(C3)$
 $\wedge \text{Plane}(P1) \wedge \text{Plane}(P2) \wedge \text{Plane}(P3)$
 $\wedge \text{Airport}(\text{JFK}) \wedge \text{Airport}(\text{SFO}) \wedge \text{Airport}(\text{ATL})$)
Goal($\text{At}(C1, \text{JFK}) \wedge \text{At}(C2, \text{SFO}) \wedge \text{At}(C3, \text{SFO})$)

Here too, Problem 2's goal can be reached using different plans, but the optimal plan length is 9 actions, one of which is shown below:

Load(C1, P1, SFO)
Load(C2, P2, JFK)
Load(C3, P3, ATL)
Fly(P1, SFO, JFK)
Fly(P2, JFK, SFO)

Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
Unload(C2, P2, SFO)
Unload(C1, P1, JFK)

- Problem 3 initial state and goal:

Init($\text{At}(\text{C1}, \text{SFO}) \wedge \text{At}(\text{C2}, \text{JFK}) \wedge \text{At}(\text{C3}, \text{ATL}) \wedge \text{At}(\text{C4}, \text{ORD})$
 $\wedge \text{At}(\text{P1}, \text{SFO}) \wedge \text{At}(\text{P2}, \text{JFK})$
 $\wedge \text{Cargo}(\text{C1}) \wedge \text{Cargo}(\text{C2}) \wedge \text{Cargo}(\text{C3}) \wedge \text{Cargo}(\text{C4})$
 $\wedge \text{Plane}(\text{P1}) \wedge \text{Plane}(\text{P2})$
 $\wedge \text{Airport}(\text{JFK}) \wedge \text{Airport}(\text{SFO}) \wedge \text{Airport}(\text{ATL}) \wedge \text{Airport}(\text{ORD})$)
Goal($\text{At}(\text{C1}, \text{JFK}) \wedge \text{At}(\text{C3}, \text{JFK}) \wedge \text{At}(\text{C2}, \text{SFO}) \wedge \text{At}(\text{C4}, \text{SFO})$)

For Problem 3, the optimal plan length is 12 actions. Here's a sample plan that is optimal:

Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P1, ATL, JFK)
Fly(P2, ORD, SFO)
Unload(C4, P2, SFO)
Unload(C3, P1, JFK)
Unload(C2, P2, SFO)
Unload(C1, P1, JFK)

After a successful test, `python run_search.py -h` is run on the command line to provide data used for the various search strategies..

Uninformed Search Strategies Analysis

For uninformed search strategies or blind search no additional information about states beyond that provided in the problem definition, here the strategies generate successors and distinguish a goal state from a non-goal state. In uninformed search strategy, the performance of seven strategies in terms of speed (execution time, measured in seconds), memory usage (measured in search node expansions) and optimality (Yes, if a solution of optimal length is found; No, otherwise). The results of the analysis is shown below.

Performance measures were collected using the following commands:

```
python run_search.py -p 1 -s 1 2 3 4 5 6 7
```

```
python run_search.py -p 2 -s 1 3 5 7
python run_search.py -p 3 -s 1 3 5 7
```

Results

Search Strategy	Optimal	Path Length	Execution Time (s)	Goal test	Node Expansions
Breadth First Search	yes	6	0.067	56	43
Breadth First Tree Search	yes	6	2.012	1459	1458
Depth First Graph Search	No	12	0.017	13	12
Depth Limited Search	No	50	0.252	271	101
Uniform Cost Search	yes	6	0.080	57	55
Recursive Best First Search	yes	6	5.841	4230	4229
Greedy Best First Graph Search	yes	6	0.011	9	7

Problem 2

Note that for Problem 2 and 3, because their execution time exceeded 10 minutes, we cancelled data collection for Breadth First Tree Search, Depth Limited Search, and Recursive Best First Search. For the same reason, with Problem 3 we did not collect any data for Breadth First Tree Search, Depth Limited Search, Uniform Cost Search, and Recursive Best First Search.

Search Strategy	Optimal	Path Length	Execution Time (s)	Goal test	Node Expansions
Breadth First Search	yes	9	25.595	4612	3346
Breadth First Tree Search	--	--	--	--	--
Depth First Graph Search	no	1082	14.519	1125	1124
Depth Limited	--	--	--	--	--

Search					
Uniform Cost Search	yes	9	22.793	4855	4853
Recursive Best First Search	--	--	--	--	--
Greedy Best First Graph Search	no	21	4.692	1000	998

Problem 3

Search Strategy	Optimal	Path Length	Execution Time (s)	Goal test	Node Expansions
Breadth First Search	yes	12	193.700	18098	1466
Breadth First Tree Search	--	--	--	--	--
Depth First Graph Search	no	596	6.1038	628	627
Depth Limited Search	--	--	--	--	--
Uniform Cost Search	yes	12	99.830	18237	18235
Recursive Best First Search	--	--	--	--	--
Greedy Best First Graph Search	no	22	30.871	5616	5614

Analysis

Breadth First Search and Uniform Cost Search are the only two uninformed search strategies that continuously produce an optimal action plan. As concerns execution speed and memory usage, Depth First Graph Search is the fastest and uses the least memory as seen in the above tables for each problem, but it does not generate an optimal action plan (problem 1: plan length of 12 instead of 6, problem 2: plan length of 1082 instead of 9, problem 3: plan length of 596 instead of 12).

Uniform Cost Search, Breadth First Search are best of the strategies consistently yielding optimal, hence the recommended search strategy. Greedy Best First Graph Search is the best alternative seeing that in problems 1 and 2, it finds the optimal path. In problem 3, it does not find the optimal path

but the path length it generates is 22 instead of 12.

Informed (Heuristic) Search Strategies Analysis

Informed method add domain specific information,add domain specific information to select what is the best path to continue searching along,Also informed search strategy is one that uses problem-specific knowledge beyond the definition of the problem itself and finds solutions more efficiently than can an uninformed strategy do. Here,the evaluation is done as above in terms of speed, memory usage and optimality.

Performance measures were collected using the following commands:

```
python run_search.py -p 1 -s 8 9 10
```

```
python run_search.py -p 2 -s 8 9 10
```

```
python run_search.py -p 3 -s 8 9
```

Problem 1

Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
A* Search with h1 heuristic	yes	6	0.080	55
A* Search with Ignore Preconditions heuristic	yes	6	0.06	41
A* Search with Level Sum heuristic	yes	6	2.620	55

Problems 2 and 3, because its execution time exceeded 10 minutes, we did not collect data for A* Search with Level Sum heuristic. For the same reason, with Problem 3 we did not collect any data for Breadth First Tree Search, Depth Limited Search, Uniform Cost Search, and Recursive Best First Search.

Problem 2

Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
A* Search with h1 heuristic	yes	9	23.807	4853
A* Search with Ignore Preconditions	yes	9	7.547	1450

heuristic				
A* Search with Level Sum heuristic	yes	9	42.353	1453

Problem 3

Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
A* Search with h1 heuristic	yes	12	100.596	18235
A* Search with Ignore Preconditions heuristic	yes	12	28.328	5040
A* Search with Level Sum heuristic	--	--	--	--

Analysis

All the heuristic strategies yield optimal action plan.h1 and Ignore Preconditions heuristics return results within the 10mins max execution time. *A* Search with Ignore Preconditions heuristic is the fastest as seen in all the above problems. If we let search run to completion on our machine, A Search with Level Sum heuristic uses the least memory*, but its execution time is much slower (26 mn for problem 2!)*

Informed vs Uninformed Search Strategies

In this section I will take the best of non heuristic strategy and heuristic strategy search that generate optimal plan actions. As explain above under uninformed search strategies that **Depth First Graph Search** is faster and uses less memory than Uniform Cost Search. As for informed search strategies, *A Search with Ignore Preconditions heuristic** is the fastest and uses the least memory.The results is bellow:

Problem 1

Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
Breadth First Search	yes	6	0.652	43
A* Search with Ignore	yes	6	0.0593	41

Preconditions heuristic				
-------------------------	--	--	--	--

Problem 2

Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
Breadth First Search	yes	9	26.718	3349
A* Search with Ignore Preconditions heuristic	yes	9	7.190	1450

Problem 3

Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
Breadth First Search	yes	12	190.753	14663
A* Search with Ignore Preconditions heuristic	yes	12	28.812	5640

*A Search with Ignore Preconditions heuristic** would be the best choice overall choice heuristic for solving Air Cargo problem because of its fastness that override the others and less memory.

Conclusion

From our experiments and results from using informed search strategies and uninformed search strategies for finding optimal plan action, *A Search with Ignore Preconditions heuristic is the best of informed search strategy and the best of uninformed search strategy because they take less time(fastness) and enough memory usage . Overall there is a best among all the strategies which is from informed search strategy and its A Search with Ignore Preconditions heuristic with the advantages in terms of speed and memory usage.*

